2012

A Behavior Based Covert Channel within Anti-Virus Updates

D. Anthony

D. Johnson

P. Lutz

B. Yuan

A Behavior Based Covert Channel within Anti-Virus Updates

D. Anthony, D. Johnson, P. Lutz and B. Yuan
Department of Networking, Security and Systems Administration, Rochester Institute of Technology, Rochester, New York, USA

Abstract - *This paper presents a new behavior based covert channel utilizing the database update mechanism of anti-virus software. It is highly covert due to unattended, frequent, automatic signature database update operations performed by the software. The design of the covert channel is described; its properties are discussed and demonstrated by a reference implementation. This paper uses these points to strengthen the inclusion of behavior-based covert channels within standard covert channel taxonomy.*

Keywords: Covert Channels, Security, Data Hiding

1 Introduction

B.W. Lampson [2] originally defined covert channels as communication channels not intended for communication at all. Covert channels are typically categorized into one of two traditionally accepted definitions. Storage based covert channels are described as any direct (or otherwise) writing of information value(s) into a legitimate overt channel by a sender, and the direct (or otherwise) reading of those information value(s) by a receiver [3]. Timing based covert channels are channels in which the message sender relays information via the modulation of resources such as CPU usage, or modulating the arrival of information such as network packets over time which allows a receiver who also knows the modulation method used to decode the message [3]. These two definitions adequately describe the vast majority of covert channels; however this paper aims to strengthen the argument for the inclusion of a third classification standard for covert channels.

2 Previous Covert Channels

There have been multiple covert channels designed and implemented by researchers in the past which fit the standard Storage or Timing based definitions. Rowland describes how small messages could be sent during the initial 3-way handshake of TCP connections[9]. Specifically, the initial sequence number could be modified to relay data covertly. The authors in [10] examine 22 potential network storage covert channels within IPv6 headers. Okamura and Oyama discovered a scenario in which two distinct and isolated virtual machines running on the Xen Hypervisor may be able to transmit covert messages between each other through the timing modulation of CPU loads [8].

3 Behavior Based Covert Channels

Behavior based covert channels are a relatively recent development in the covert channels taxonomy. Johnson, Lutz and Yuan [4] explain how the purposeful alteration of the internal states or behavior of an application can allow the leakage of information between two parties [4]. A game could be used as a covert channel if two parties agree upon a handshake initialization as well as protocol for encoding a message through moves made within the game [4]. The authors use a game called Magneton to demonstrate the channel. This behavior based model needs to be recognized as a completely separate method of classifying a covert channel, and can be used to uniquely identify the Anti-Virus Covert Channel (AVCC, a new covert channel presented in this paper). AVCC contains qualities that could be neither described as storage based or timing based alone. Instead, it is the *behavior* of the anti-virus scanning engine after a database update has been applied which will allow a covert message to be transmitted.

4 Signature Based Anti-Virus System

To understand how an Anti-Virus Covert Channel would function, Anti-Virus signatures must be discussed. Anti-Virus signatures typically will take the form of a set of unique cryptographic hashes or pieces of code which corresponds to a known malicious file[6]. These signatures will typically reside within a database that an Anti-Virus application will use to identify known malware that resides on a system. If a match is discovered, the Anti-Virus application will alert the user that a malicious file has been identified and quarantine the file. These Anti-Virus databases must be kept up-to-date to ensure virus scans are accurate and effective. There are many Anti-Virus products on the market and each will have their own method of updating the signature database. ClamAV, a popular free and open source Anti-Virus application, makes use of a tool called Freshclam for this task. Freshclam is a highly customizable tool which may be run interactively from a command line shell or alternatively run as a daemon process on the host machine [7]. Freshclam can also be configured to automatically check for new updates to the

signature database as many as 50 times per day [5]. ClamAV provides additional security metrics such as digital signatures on all updates to be distributed. Updates to the any signature database would be considered normal network traffic and not arouse suspicion from security administrators. New malware is discovered daily and generates constant pressure for AV Companies to identify the latest threats. Database updates occur on a regular basis and may contain a large amount of information depending on the specific Anti-Virus application. For these reasons, the updating of an Anti-Virus database may be seen as an ideal method for transmitting a covert message.

5 A New Covert Channel

The Anti-Virus Covert Channel (AVCC) can be classified as an exemplary layer 7 or application layer behavior-based covert channel which can be used to subvert the existing security policy of an organization. ClamAV was chosen as the example software to demonstrate this new covert channel due to its open source, cross-platform nature combined with its user friendly signature database. It should be noted however, the design of this covert channel is not specific to a certain brand of Anti-Virus, and the ideas presented could theoretically be applied to any AV implementation.

The Prisoner's Problem mentioned in [1] is used to describe the method in which this covert channel may be applied. One node on the internet will be sending data to a second node while attempting to prevent this communication from detection by a third party. The sender will have access to either an official anti-virus signature distribution point or own a location that can be used for updates. The sender will then encode specially crafted signatures into updates which will later be used to relay a message to the receiver. Each signature added into the update will be representative of a binary "1" within the covert message. At designated intervals the receiver will update his/her anti-virus signatures which will force the retrieval of the database update crafted by the original sender node. The receiver will then scan a unique directory of files with his/her updated anti-virus application. The results of the directory scan will be used to decode the covert message. It is this *behavior* of the anti-virus program during the scan, not solely the signatures within the update which relay the covert message. This flow of data transfer is pictured in Figure 1. A proof-of- concept implementation will be discussed later in this paper.

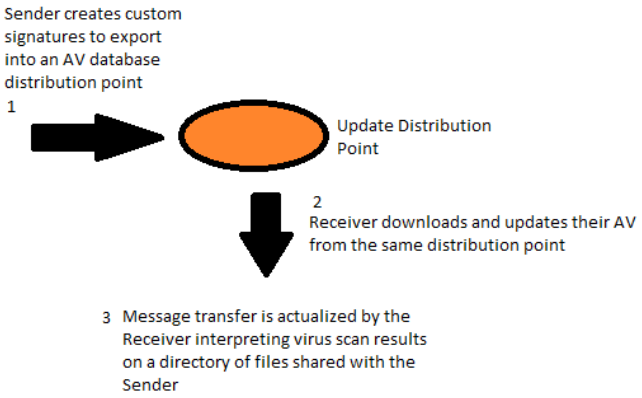

Figure 1. Data Transfer

5.1 Covertness

The transferred covert message is never fully contained within the updates which precludes classification as a strictly storage based covert channel. Only binary "1's" from the message are transferred through the update mechanism- not "0's". Even if a third party was able to capture the network traffic and knew how the channel operated, the covert message would be unable to be revealed from physical inspection. Additionally, this covert channel is not based upon the modulation of timed events which implies that it is not a timing based covert channel. If a third party is able to observe the timing of all network events, no irregularity can be detected as AV signature updates are a normal regular network event and in some cases may be required by an organization's security policy.

5.2 Data Rate

Anti-virus database updates vary in size depending upon the specific application being discussed. These updates are often based upon the discovery of new pieces of malware which varies and can have an impact on the amount of data contained within an update. Some days may have more information stored within an update than others. There is no theoretical limit to the size of a signature database. In practice, each specific anti-virus product will have an average update size. The data rate of the AVCC will be closely tied to the specific anti-virus application which is chosen for covert channel use. Covert messages transferred using the AVCC will also have different data rates depending upon the type of data being sent. According to channel design, instances of a binary "1" within a message will require a signature being added to a database update. To remain covert, the modified database updates must remain similar in size to normal updates. It may arouse suspicion if certain updates are exceedingly large. Since each binary "1" within a covert message correlates to a new signature being created, the data rate of the AVCC may be measured by the number of signatures that can be added while limiting the size characteristic to remain within the confines of a normal database update. Once this limit has been determined, a node

is only limited to sending messages containing binary "1's" which fall below that threshold.

5.3 Robustness

If packets do not arrive in a specific order or network timing is disrupted, the integrity of a timing based covert channel can suffer; many storage based covert channels do not survive packet regeneration performed by proxies or routers. The AVCC design does not succumb to these types of obstacles. Update mechanisms of application layer software will typically make use of the Transmission Control Protocol (TCP) to insure reliable data delivery. Two ways in which a covert message could be modified or tampered with is the complete prevention of data transfer or removal of a sender's specially crafted signatures before the transmission. If the data sent in an update maintains integrity and reaches the receiver, the message will be relayed. This gives the AVCC a high score in the robustness category.

5.4 Limitations

The primary concern of the AVCC design is the issue of a sender controlling a database update location. It may not be ideal or feasible in certain situations for a user to be an operator or have access to locations in which database updates can be distributed. In cases regarding proprietary software this access becomes even more difficult. Another limitation that should be noted is that the type of data being transferred will make a significant impact on the data rate. As each binary "1" in a message corresponds to a virus signature that must be added, data made up of many "1s" will increase the size of updates to a greater degree. Staying inconspicuous requires staying beneath an update size threshold, thus will have an impact on the data throughput of the covert channel. A minor area of concern could be the simplex style of communication signature updates provide, however this same characteristic also makes AVCC a good choice for the infiltration of messages or data into an organization.

6 Assumptions

Before a Proof of Concept was created and tested, the following assumptions are applied. The model assumed in this paper precludes a third party from physical tampering with the machines involved in the covert message sending/receiving, including modification of information during transit. This covert channel does not currently offer the security characteristic of authentication or data integrity- simply transfer of covert messages. It is also assumed that persons wishing to use the AVCC will be able to access the distribution site for ClamAV or another anti-virus and be able to direct their host AV software to receive updates from this same location. This will also attempt to accurately represent the use of this covert channel in the real world where network defense/defenders would not initially be aware of AVCC's existence.

7 Proof of Concept

7.1 Software Design

The AVCC has been created and implemented in Python as a proof of concept. The implementation script (avcc.py) is a "wrapper" to the tools and binaries that ClamAV provides with a default installation. The creation of signatures is managed with *sigtool* which is the custom signature and database management tool that accompanies ClamAV. Sigtool will allow senders to easily create and add signatures which avcc.py uses to automatically encode/decode the covert message. Alice and Bob have a shared secret which takes the form of a directory of files which will henceforth be referred to as keyfiles. They each share the exact same number, order, and md5sum property for the directory of keyfiles. These keyfiles are what will be scanned with ClamAV when the update has been retrieved. This proof of concept makes use of ASCII encoded messages for all covert communication. The avcc.py script contains methods for the encoding of a message, decoding of a message, and the creation of unique keyfiles based upon a single unique executable.

7.2 Example Signatures

The avcc.py implementation of AVCC works by creating signatures for specific files in the keyfiles directory. ClamAV updates can make use of many different types of databases, however this proof of concept will specifically make use of the .hdb database which defines MD5 based signatures. All signatures in this database refer to an executable file.

This database contains the most basic form of an Anti-Virus signature that ClamAV implements and is perfect for the proof-of-concept due to simplicity and ease of creation. Figure 2 displays example signatures that have been added to the database by the avcc.py script.

```
b5bd110baa6139b011c21d713644b031:7099:keyfiles/keyfile.12
75a5e7d3def5bc68adff0817dd731cbd:7099:keyfiles/keyfile.17
6bc95a1830db073cdcc9fb608d6c21d5:7099:keyfiles/keyfile.18
47f0763ef540813c61e436d47b123c69:7099:keyfiles/keyfile.20
8acd9bb624b0bf4782cac67337ad1e69:7099:keyfiles/keyfile.22
f1e4690e24731bd779ba19a2cc78e6f7:7099:keyfiles/keyfile.24
4e14c80d6e302f475ce21b090dd36e51:7099:keyfiles/keyfile.25
bbd11c02a1427a31e9e5a59def5cbf4e:7099:keyfiles/keyfile.27
e2b39311a68d333555961367e930bd7d:7099:keyfiles/keyfile.28
2a46a262f3de33a7b3efbb2cd08b4f85:7099:keyfiles/keyfile.31
```

Figure 2. Example Signatures

These virus signatures are made up of three colon delimited fields. The first field is the MD5 sum of the contents of a file. The second and third sections are the size of the file and its name (given by sigtool) respectively. As it may not be practical to have an identical directory of unique executable files on both ends of transmission, the proof-of-concept is able to generate a keyfiles directory. Keyfile creation with avcc.py is based upon the modification of a single shared executable file. This allows the sender and receiver to produce the same set of keyfiles from one original

executable file. This also prevents the need to send a shared directory of keyfiles beforehand and potentially arouse suspicion. In real world scenarios, sender and receiver could arbitrarily choose an executable to generate the keyfiles from. They may even decide to use a commonly shared file on their machines and prevent the need of transferring a file at all. During keyfile generation avcc.py modifies the contents of the base executable slightly to ensure uniqueness between keyfiles. The example keyfile directory shown in Figure 2 contains files named in the fashion *keyfile.number*.

7.3 Encoding

Encoding of the message can be demonstrated by the following pseudo code example:

- Populate a keyfile array with keyfile names
- Accept User Input (string of chars)
- Translate input into binary array
- Iterate through binary array
- If (current value == 0)
** - Do nothing**
- If (current value == 1)
** - Create signature for the keyfile in the corresponding element of keyfile array**
- Distribute created signatures

Avcc.py has been created to accept user input in the form of an ASCII string. The string will then be converted into a binary array. The keyfiles directory will be examined and an array will be created. Each entry within the keyfile array contains the name of a keyfile. The binary array will be examined and encoding takes place based upon this array. If the current entry in the binary array is a 0, that means that no signature will be created for the file in the keyfile array. If the current entry is a 1, then a signature is required to be added to the database update. This encoding method will create 1 signature to be added to the .hdb database file contained within a ClamAV update for every binary "1" of the data being transferred.

7.4 Decoding

Decoding is the reverse of the encoding process. When the receiver has updated his database of signatures from the sender, a virus scan is run upon receiver's directory of keyfiles. The decoding can be explained via the following pseudo code example:

- Download database update
- Create empty array
- Create keyfile array from names of keyfiles
- Iterate through keyfile array
- Scan current element
- If (Virus Detected)
** - Push "1" onto empty array**
- If (No Virus Detected)
** - Push "0" onto empty array**
- Convert binary array into real binary data representation
- Print binary data representation as string
- Message is revealed

Avcc.py has been created to automatically reveal the hidden message within a signature database update. The receiver's directory of keyfiles will be scanned in order with ClamAV. Every keyfile that triggers a virus-alert will be represented as a binary "1" and keyfiles with no detection are represented as a "0". This sequence of 1s and 0s is representative of the binary data being transferred. Avcc.py will keep track ClamAV behavior and this order-specific sequence of bits and reveal the hidden message to the receiver.

7.5 Performance

A quantitative analysis of avcc.py performance was done with the assistance of Nathaniel Morefield using the Minitab statistics software. Avcc.py uses ASCII encoded strings for each covert communication. Representations of ASCII characters contain an average of 3.5 binary "1s". Over the course of 5 weeks, 37 ClamAV database updates were collected from which conclusions about update size were drawn. The mean size of the .hdb file contained within a ClamAV update was 30,229 bytes and contained a mean of 503 unique signatures. In order to keep the additions to the update inconspicuous, total update size *including the message* should be no more than two standard deviations from the mean. The standard deviation of this set of data was 5,950 bytes. This allows for an additional 11,900 bytes of data added to an average update using the assumptions above. A regression test was performed to determine the average size of a single virus signature. The test was restricted so that the best fit line passed through the origin, since an update with no virus signatures would have a size of zero bytes.

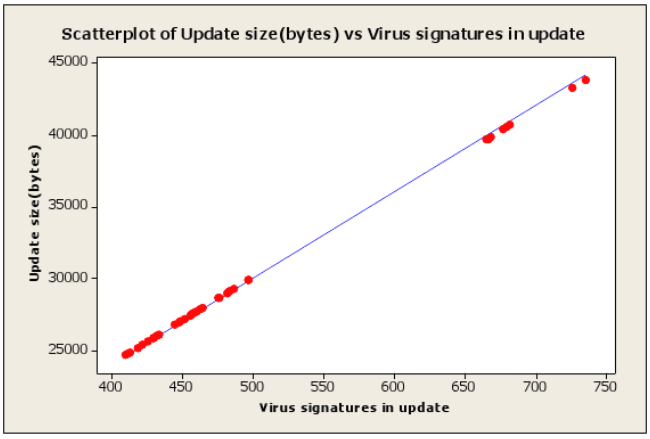

Figure 3. Regression Analysis

Figure 3 shows the scatter plot and regression line associated with the sample data. From this test it has been determined that the average virus signature requires 60.1 bytes of data. Each ASCII character is represented with an

average of 3.5 binary "1s"; it will require 210.35 bytes to encode one ASCII character. In order to stay within the two standard deviation size limit of 11,900 bytes, approximately 57 ASCII characters can be effectively hidden within the covert channel.

7.6 Environment

The environment that avcc.py has been tested on is Ubuntu 10.04 x86 Server Edition. This platform provided an environment that was stable and easy to work with.

7.7 Limitations of Implementation

The implementation avcc.py is a proof of concept of how an anti-virus database update could be used as a covert channel. A fully-functional implementation would require a server that both digitally signs and packs many different types of anti-virus signatures into one file for distribution. This proof of concept strictly details how one may encode covert data within a .hdb. The .hdb file is just one of many specific types of databases that are packed together and digitally signed for a ClamAV signature update. The implementation could be extended to any of the other types of databases, however this proof of concept is not that ambitious. If the methods described in this paper were applied to a fully functional anti-virus signature database distribution point, this covert channel should operate flawlessly.

7.8 Detection and Prevention

The implementation of avcc.py is used with unofficial signatures for testing purposes. The files that are used for the creation of signatures are recognized as unofficial signatures by the ClamAV scanning product. The finding of many unofficial signatures could be one way in which anything suspicious could be noted outside of a complete audit of every signature contained within update. Auditing each file in a regular update would be theoretically possible although unfeasible in practice, so this covert channel would be very difficult to detect. A method of prevention could be to audit the locations that a user is updating their signature database from. Restricting update location to approved locations would be one way in which this covert channel could be prevented altogether. This prevention method would be futile if a sender was able to gain access to an official update location. Another method that a sender could use to thwart detection would be to submit new legitimate virus signatures to the official website, which would inevitably be added to the next official update. These "real" signatures could also then be used to transfer a covert message.

8 Conclusions

In the future, we would like to explore new possibilities of sending covert communication relating to software updates and further the support for behavior based covert channels being included in standard taxonomy. Anti-Virus software tends to fly under the radar of security programs that monitor for malicious activity. Anti-Virus products are some of the most popular types of applications in use today, and if this technology can be harnessed for use as a covert channel it may prove to be a rich area of further research.

9 References

[1] Simmons, G. J. "The Prisoner's Problem and the Subliminal Channel", Advances in Cryptology: *Proceedings of* CRYPTO '83, Plenum Press, 1984, pp. 51-67.

[2] B. W. Lampson. "A note on the confinement problem".*Communications of the ACM*, 16(10) pp. 613-615, 1973.

[3] Zander, S., Armitage, G. and Branch, P, "A survey of covert channels and countermeasures in computer network protocols", *IEEE Communications Surveys & Tutorials,* 9(3) pp. 44-57, 2007.

[4] Johnson, D. and Lutz, P. and Yuan, B. "Behavior-based covert channel in cyberspace", in: Vanhoof, et al (eds), *Intelligent Decision Making Systems*, World Scientific, 2009, pp. 311-318.

[5] Kojm, Tomasz. "Freshclam(1) – Linux man page". Internet:http://linux.die.net/man/1/freshclam , Jan. 23, 2012.

[6] Landesman, Mary. "What is a Virus Signature?". Internet:http://antivirus.about.com/od/whatisavirus/a/virussignature.htm, November 29, 2011

[7] Kojm, Tomasz. "Clam AntiVirus 0.97.4 User Manual". Internet:http://www.clamav.net/doc/latest/clamdoc.pdf , 2007-2011. Jan 17, 2012

[8] Okamura, Keisuke and Yoshihiro Oyama. "Load-based Covert Channels between Xen Virtual Machines." In *Proceedings of the 2010 ACM Symposium on Applied Computing*. Pp 173-180. 2010.

[9] C. H. Rowland, "Covert channels in the TCP/IP protocol suite." *First Monday,* vol. 2,no. 5, 1997

[10] Lucena, N., Lewandowski, G., and Chapin, S. "Covert Channels in IPv6", *Lecture Notes in Computer Science (2006)*. Vol. 3856, Springer, Pages 147-166